APL AS ROSETTA STONE LANGUAGE

Edited by
Toshio Nishikawa Dr. Sci.

PREPRINT

SCIENCE HOUSE

APL AS ROSETTA STONE LANGUAGE

Edited by
Toshio Nishikawa Dr. Sci.
National Chemical Laboratory for Industry
AIST, MITI, Japan

PREPRINT

SCIENCE HOUSE CO., Ltd.
Tokyo, Japan

Contributed by

English & Japanese
Toshio Nishikawa
National Chemical Laboratory for Industry
Tsukuba Research Center, Japan.
English
Thomas M.Olsen
Karsten Manufacturing Corp.
Phoenix, Arizona, USA
Leroy J.Dickey
University of Waterloo
Ontario, Canada
German
Traude Banex Freihalter
APL UG Germany,
West Germany
French
Valerie Tardif
University of Waterloo
Ontario, Canada
Chinese
Jinpei Zhang
Nankai University
Tianjin, China

Edited by
Toshio Nishikawa
National Chemical Laboratory for Industry
Tsukuba Research Center, Japan.

Published by
Science House Co., Ltd.
Tokyo, Japan

APL as Rosetta Stone Language

by Toshio Nishikawa
National Chemical Laboratory for Industry
Tsukuba Research Center, 305, Japan.
Printed in Japan
by Science House Co., Ltd
BY House 3F, Yusima 1-9-7, Bunkyo-ku,
Tokyo, 113, Japan

Preface

APL (A Programming Language), first invented by Dr.Iverson, is noted as both unique and productive programming language. It uses special symbols called APL characters, while almost all other programming languages use keywords derived from the one natural language, English. In this sense, APL would be considered as unbiased and truly international language. It could be compared to Esperanto by Dr.Zamenhof, or further to universal mathematical symbols. It is certainly reasonable that APL comes from the original idea of Dr.Iverson, a mathematician.

This booklet, entitled as 'APL as Rosetta Stone Language' is a natural language comparison based on APL simple programming examples.

The idea was stimulated by an inspiration at the APL 88 Conference, Sydney; some people said to me that our Japanese APL book would be the most convenient for APLers to learn the Japanese language itself.

After the Conference, I requested to some APLers over the world, who may write the text in his or her native language. This may well be called as a 'Rosetta Stone' based on APL programming.

The original text adopted here is taken from the extract of 'The First Step of APL —— APL Beginner's Primer for RIPS Users at AIST, Tsukuba Research Center, Japan written in Japanese by the author.

So far, I've collected five different language versions: English, German, French, Japanese and Chinese. A little change of the text for editorial is solely responsible to me.

I believe this booklet will be helpful to understand each other between alien people as well as to learn APL programming.

I would like to thank the APLers listed, who contribute in their own native languages: Tom Olsen, Traude Banex, Lee Dickey, Valerie Tardif and Jinpei Zhang. Besides, I express thanks to Yasuhiro Iihashi and Masahiro Doi, Science House Co., Ltd. for troublesome typography and layout of multi-lingual texts. Never it would be fulfiled without their kind cooperations.

Toshio Nishikawa

This booklet is submitted to the presentation at the APL 89 Conference, New York.

What is APL?

Was ist APL? **Qu'est ce qu' APL?**

APL とは何か？ **APL是什么？**

```
**********************************
*                                *
* APL IS HAPPY APL IS HAPPY A *
* PL IS HAPPY APL IS HAPPY AP *
* L IS HAPPY APL IS HAPPY APL *
*  IS HAPPY APL IS HAPPY APL  *
* IS HAPPY APL IS HAPPY APL I *
* S HAPPY APL IS HAPPY APL IS *
*  HAPPY APL IS HAPPY APL IS  *
* HAPPY APL IS HAPPY APL IS H *
* APPY APL IS HAPPY APL IS HA *
* PPY APL IS HAPPY APL IS HAP *
* PY APL IS HAPPY APL IS HAPP *
* Y APL IS HAPPY APL IS HAPPY *
*                                *
**********************************
```

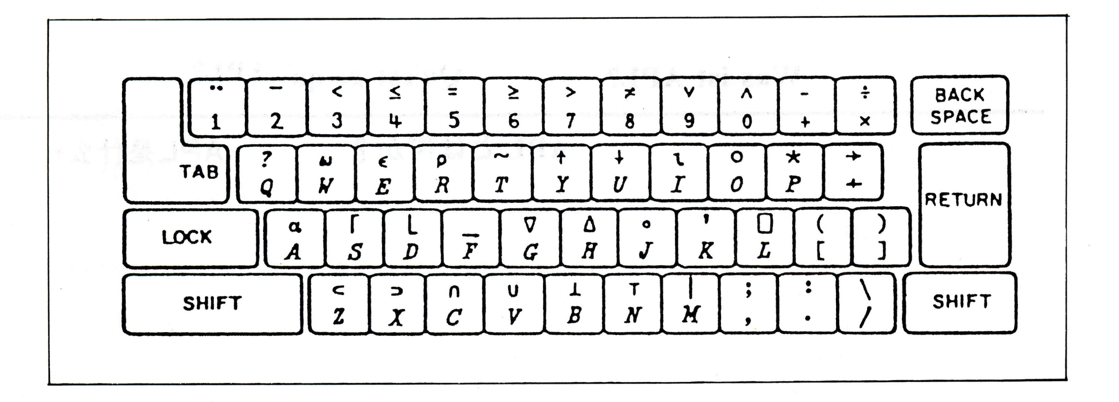
We can easily get the pattern on our display terminal.
Can you do it?

Dieses Muster können wir ganz einfach auf unserem Bildschirm erzeugen.
Können Sie das nicht?

Il est facile d'obtenir cette image sur votre écran.
Voulez-vous essayer?

このパターンを簡単にディスプレイ画面に出すことができます。
さあ，どうしたらよいでしょう？

我们能在显示终端上轻易得到上面的输出，你能做到吗？



How to use APL simply

APL (A Programming Language) is an interactive programming language, which may powerfully act in matrix form for all kinds of data processing.

By just entering a few lines, you will get the pattern shown on the first page.

APL uses special characters shown in the keyboard. However, you need not shrink back, because you are not familiar with it. Just like using a pocket calculator, you pick up a key, then push it. For example, you need not enter the word

STORE

but just push the key

[←]

That is, if you want to operate APL, you need not spell out language words.

Wie Sie APL qanz einfach anwenden

APL (A Programming Language) ist eine interaktive Programmiersprache für alle Arten der Datenverarbeitung, die sich aufgrund der matrixorientierten Arbeitsweise als besonders mächtig erweist.

Das auf der ersten Seite gezeigte Muster erhalten Sie durch Eingabe weniger Zeilen.

APL verwendet besondere Zeichen, die auf der abgebildeten Tastatur dargestellt sind. Sie müssen jetzt nicht erschrecken, nur weil Ihnen diese Zeichen fremd vorkommen. Ebenso, wie Sie einen Taschenrechner benutzen wählen sie eine Taste und drücken sie. Sie brauchen keine Worte einzugeben, wie zum Beispiel

SPEICHERN

statt dessen drücken sie einfach die Taste

[←]

Das heisst, wenn Sie mit APL arbeiten brauchen Sie keine Worte in irgend einer Sprache einzutippen.

Comment utiliser facilement APL?

APL (A programming Language) est un langage de programmation interactif, qui peut ŕeussir toutes sortes d'opérations puissantes à l'aide de matrices. Il faut seulement quelques lignes pour obtenir l'image du début. Essayez par vous-même.

APL utilise des caractères bien particuliers comme ceux du clavier illustré ci-bas. Toutefois, ne vous laissez pas décourager par leur aspect peu familiar. Comme une calculatrice de poche, il s'agit seulement de presser la clé. Por exemple, vous n'aurez pas à entrer le most

STORE

mais seulement presser la clé

[←]

Avec APL, vous n'aurez jamais à épeler les mots tout au long.

APLの簡単な使い方

APL (A Programming Language) は配列によるデータ処理を得意とする会話型言語です.

最初のページのパターンはほんの2,3行を入力するだけで手軽にできます.

APLでは、キーボードに示すような特別の文字を使います。しかし、これを見慣れないとは思わないで下さい。電卓と同じように、必要なキーをさがして1本指で打てばよいのです。例えば

STORE

と打つかわりに

[←]

と打ちさえすればよいからです。

つまり、APLでは他の言語の場合のようにふつうの「タイピング」の感覚で単語を入力する必要はありません。

APL的简单用法

APL(A Programming Language)是会话式的程序语言，在处理数组以及所有形式的数据上有很强的能力。

只要输入简单的几行命令就能得到如前页所示的输出。

(APL键盘上使用了一些特殊字。)但是,别因为不熟悉这些特殊字而退却。(就象使用计算器那样，找到需要的键按一下即可。比如：想打

STORE

时，只要打一下

[←]

键就可以了。

也就是说，在运用APL时,不需要象使用其它语言那样输入单词。

```
LINE     1       COLUMN 1

         V S  A P L   4.0

CLEAR WS

         23 + 35
58

         23 - 35
¯12

         23 + ¯35
¯12

         ¯35 + 23
¯12

         3.2 × 12
38.4

         32 ÷ 5
6.4

         2 * 4
16

         10 * 0.5
3.16227766

         2 * 0.5
1.414213562
```

Let's use an APL system like a pocket calculator. You may invoke an input into the APL system at the seventh column as is indicated by the prompt cursor. It is not prompted by e.g. 'READY'. Thus, you may enter any statement, character by character, at that position. You'd better put a 'blank-space' to be clearly readable, although it is not necessary for the language specification. You end each statement by an 'enter'-key. Immediately afterwards, the system returns calculated results beginning at the first column of the next line.

Please try further examples. APL uses an over-bar (not hyphen) as a minus sign for a negative number instead of hyphen(−), to be discriminated from subtraction. At a glance, since subtraction is an operation between two numbers, while a minus sign denotes the nature of a number, you will see such a method is more rational. The following two statements work in the same way.

Multiplication is performed frankly with '×', instead of ' * '. Numbers with decimal point and integers may be freely mixed in arithmetic calculation. Division is also performed with '÷' instead of '/'. Power operation is done with ' * '(asterisk) and enables decimal fraction: e.g. the power of 0.5 equals the square root.

"APL is a language with easy access like a pocket calculator."
——Enter data as it is, and then you get the result immediately.

Zunächst wollen wir APL wie einen Taschenrechner nutzen. Sie können Ihre Eingabe an das APL System dort machen, wo der Cursor steht, das ist meist in Spalte 7. Sie werden durch keinen Befehl, wie z.B. 'READY' zur Eingabe aufgefordert. Das heisst, dort wo der Cursor steht können Sie jede beliebige Anweisung eingeben, Zeichen für Zeichen. Wegen der besseren Lesbarkeit können Sie Leerstellen einfügen, dies wird jedoch von der Sprache nicht gefordert. Ist eine Anweisung fertig eingegeben, drücken Sie die ENTER-Taste. Sofort gibt das System die errechneten Ergebnisse zurück. Es beginnt damit in der ersten Spalte der nächsten Zeile.

Bitte probieren Sie weitere Beispiele aus.

APL benutzt anstelle eines Bindestrich ein hochgestelltes Minus als Kennzeichen für eine negative Zahl, um den Unterschied zur Subtraktion sichtbar zu machen. Kurz gesagt, da eine Substraktion eine Rechenoperation zwischen zwei Zahlen darstellt, ein Minuszeichen jedoch den Zustand einer Zahl angibt, werden Sie diese Methode vernünftiger finden. Die folgenden beiden Anweisungen arbeiten auf diese Weise.

Multiplikationen werden einfach mit '×' durchgeführt, anstelle von ' * ' Dezimal- und Integer-Zahlen können in arithmetischen Ausdrücken beliebig gemischt werden. Divisionen werden mit '÷' durchgeführt, anstelle von '/'.

"APL ist eine Sprache einfach wie ein Tascherechner."
—— Sie können die Datem benutzen sofort und danm die Ergebniss wird durch gebührt.

Le systeme APL s'utilise comme une calculatrice de poche. Comme indiqué par le curseur, les entrées se font à partir de la septième colonne. Vous ne verrez jamais de signal READY. Vous entrez votre commande, caractère par caractère, en insérant des espaces blancs pour faciliter votre compréhension si vous le désirez. Chaque ligne doit se terminer en pressant la clé ENTER et le système renvoie immédiatement les résultats sur la première colonnede la ligne suivante.

Voici plusieurs exemples que vous pouvez essayer.

Comme vous pouvez le remarquer, APL utilise la barre haute (à ne pas confondre avec le trait d'union) pour indiquer la négation, alors que la soustraction (−) est considérée comme une opération binaire qui requiert deux arguments. Cette méthode permet de faire une distinction à première vue entre les deux opérations. Ceci est illustré par les deux lignes suirantes de notre exemple.

La multiplication est définie par un × plutôt que par une étoile (*). La division, par un ÷, plutôt que par une /, comme on le voit souvent. Les nombres décimaux et les entiers peuvent être librement utilisés à l'aide de la même commande arithmétique. Les puissances sont indiquées par une étoile et permettent l'utilisation de fractions. La puissance 0.5 correspond à la racine carrée.

"APL est un langage facile d'accès."
—— On l'utilise comme une calculatrice de poche, en y entrant l'information telle quelle. Les résultats apparaissent immédiatement devant vous.

APLを電卓と同じように使って、システムと会話してみましょう。

APLでシステムへの入力は、カーソルにより示されるように、7桁目から行います。他の言語のように、例えばREADYといったプロンプトでは示されません。したがってそこから一字ずつ打ち込んでいきます。区切りのスペースは、ここでは入れなくてもかまいませんが、見やすくするためには、入れた方が良いでしょう。最後に ENTER キーを入れます。するとシステムからの応答である計算の答は、次の行の一番左の1桁目から表示されます。

さらにいろいろとやってみましょう。

APLでは、マイナスの符号は、オーバーバー(ハイフンとはちがう)で示され、引き算の記号-(ハイフンと同じ)とは区別されます。考えてみれば、引き算は前の数値から後の数値を引くという演算操作を示すものですが、一方マイナスの符号は、数値の性質を示すものだからです。このように区別した方が正確で、ずっと合理的です。

したがってその次の2つのAPLのステートメントは、いずれも同じです。

掛け算はすなおに×を用います。他の言語のように＊ではありません。
また、このようにすべての四則演算において小数点を持つ数と整数とが混在してもかまいません。

割り算はやはり÷を用います。/ではありません。

べき乗は、今度は＊(アスタリスク)で行います。小数のべき乗も可能で、当然ながら 0.5乗は平方根になります。

"APLは、電卓なみに手軽に使える言語である"
――データはそのまま入れればよいし、結果は即、表示される。

APL系统的用法和计算器相以。现在让我们试着与系统进行对话。在光标所示的第七列开始输入命令。这里没有任何提示(如'READY'等)，你可以在光标所在位置一个字一个字地输入任何语句。为了使语句有清楚的可读性，最好加入一些空格，但语言本身并不要求使用空格。'enter' 键表示一个语句的结束按下此键后系统立刻从下一行最左边的第一列开始显示计算结果。

请试运行更多的例子。

APL用上横杠(over-bar)作为负号，这样做是为了和减号相区别。因为减是两个数之间的运算而负表示一个数的属性，这样做更合理一些。

下面的两个语句作用相同。

乘用'×'表示，而不用'＊'。小数点数和整数在所有四则运算中可以随便混用。除用'÷'而不用'／'表示。

幂用'＊'表示，允许使用拾进制小数作为乘幂，如0.5次方等于平方根。

"APL是一种象计算器一样简单易用的语言。直接输入数据就会立刻得到结果。"

```
      X←123.45
      Y←¯0.25
      X
123.45
      Y
¯0.25
      X + Y
123.2
      X - Y
123.7
      X × Y
¯30.8625
      X ÷ Y
¯493.8
      X × 3
370.35
      Y × 3
¯0.75
      N←3
      N
3
      X × N
370.35
      Y × N
¯0.75
```

Next, let's go on with named variables like on a little more expensive calculator with memories.

A sign of left-arrow (←) works so as to assign variables X or Y to number values. Be careful that an equal sign (=) in APL does not mean assignment like BASIC etc. Using variables X and Y, you may do several calculations. Mixed calculation with variable and constant may be done.

The same result comes even if you use a variable instead of 3.

The name of a variable is free from any limitation whether it is used as an integer or as a real number. Type of number is determined at the entry of input.

"APL works freely in integer or real number"
—— No need for type declaration

Nun wollen wir mit den Namensvariablen fortfahren, das ist etwa so, als würden wir einen etwas teureren Taschenrechner mit Speicher benutzen.

Mit dem Zeichen ← 'Pfeil nach links' weisen wir hier den Variablen X und Y numerische Werte zu. Denken Sie daran, dass ein 'Ist-Gleich' -Zeichen (=) in APL keine Zuweisung bedeutet, wie z.B. in BASIC usw.

Mit den Variablen X und Y können Sie nun diverse Berechnungen durchführen. Auch Rechnungen, in denen Sie sowohl Variable als auch Konstante verwenden, können Sie durchführen.

Sie erhalten dasselbe Resultat, wenn Sie eine Variable anstelle der Konstanten 3 benutzen.

Der Name einer Variablen unterliegt keiner Beschränkung, egal ob Integer- oder Real-Zahlen zugewiesen werden. Der Zahlentyp wird zum Zeitpunkt der Zuweisung festgestellt.

"APL arbeitet ohne jede Beschränkung mit Integer oder Real-Zahlen"
—— Jede Typ-Deklaration entfallt.

Maintenant, passons aux variables qui requièrent un peu plus de mémoire, imitant une calculatrice de poche un peu plus dispendieuse.

La flèche vers la gauche (←) initialise les variables, tels que X ou Y. Prenez note que le signe =, en APL, ne signifie pas l'initialisation, comme en BASIC. Une fois X ou Y initialisée, la variable peut être plusieurs fois utilisée même avec une constante. L'utilisation de la variable initialisée à 3, ou de la constante 3, donnera le même résultat.

Le nom donné à une variable reste entièrement votre choix, quelque soit le type de la variable, qui est lui, déterminé par la valeur assignée à la variable.

"APL travaille librement avec des entiers comme avec des nombres réels."
—— Aucun besoin de déclaration de type.

今度は名前をつけた変数を使って、メモリ付き電卓と同じようにやってみましょう。

記号←(左矢印)は、変数XとYとに数値を代入することを示します。APLでは、記号=(等号)はBASICのように代入の意味では使いません。注意して下さい。

変数X,Yを使っていろいろな計算ができます。

当然ながら、変数と定数とを混ぜることもできます。3を変数に代入して、それを使っても同じです。

APLでは、変数の名前について、整数、実数の区別は必要ありません。数値を入力したときに、その型がきまります。

"APLでは、整数、実数に区別はない"。
—— 変数の型宣言はいらない。

下面让我们来看一看变量的用法，这和使用带有存储的稍贵的计算器相似。

左箭头(—)表示给变量X或者Y赋值、请注意APL不用等号(=)表示赋值，这和BASIC等不同。你可以使用变量X和Y进行各种运算，也可以进行变量和常数的混合运算。使用被赋值为3的变量来代替常数3进行运算得到相同结果。

在APL中不用区别实数和整数的变量名，变量的类型由所输入的数值来定。

"APL不用区别整数和实数"
——不需定义变量的类型。

```
      Z←123.45 ¯0.25
      Z
123.45 ¯0.25
      Z × 3
370.35 ¯0.75
      V←Z × 3
      V
370.35 ¯0.75
      Z + V
493.8 ¯1

      Z[1] + V[2]
122.7
      Z[2] - V[1]
¯370.6
```

Now, how can you carry out such calculations all together simultaneously? APL is the best tool for such purpose.

Suppose a set of numbers containing two items is assigned to a variable Z. In other words, Z works as an array composed of two elements, however, it is not necessary to declare this as e.g. 'DIMENSION'. At the point when more than two values are assigned to a variable, it is automatically set as an array. So, multiplication of each item by three may be done at the same time. By the way, a single data value which appeared previously is called a scalar. Substituting the result into another array variable, you may simply add each elements of Z and W, respectively.

To treat each element of an array, you may pick out by [and] and calculate in ordinary fashion. In APL, the values may be collectively assigned, or operated on in any arithmetic calculation.

"APL treats data collectively."
——No declaration is needed for an array.

Nun, wie können sie solche Berechnungen auf einen Schlag simultan durchführen? APL ist das beste Werkzeug für diesen Zweck.

Angenommen eine Zahlenreihe, die zwei Elemente enthält, wird der Variablen Z zugewiesen. Das heisst, Z ist ein zweielementiger Vektor, dennoch, eine Deklaration wie 'Dimension' ist nicht nötig. In dem Augenblick, wo einer Variablen mehr als ein Wert zugewiesen wird, wird sie automatisch als Bereich angelegt. Die Multiplikation jedes Elements mit 3 kann jetzt gleichzeitig erfolgen. Übrigens, ein einzelnes Datenelement, wie wir es vorher gezeigt haben, wird Scalar genannt. Wenn Sie das Ergebnis einer weiteren Variablen zuweisen, können Sie einfach jedes Element von Z mit dem entsprechenden Element von W addieren.

Um mit einzelnen Elementen eines Bereichs zu arbeiten, wählen Sie diese mit [und] aus und rechnen dann in gewohnter Weise.

In APL können Werte gemeinsam zugewiesen oder in arithmetischen Berechnungen verwendet werden.

'APL bearbeitet eine Datenmenge im ganzen'
—— Eine Deklaration für einen Bereich ist nicht nötig.

Maintenant, vous pouvez résoudre facilement de tels calculs. APL est l'outil parfait pour ce genre de travail.

Imaginons qu'un ensemble de deux nombres soit affecté à la variable Z. Z devient donc un étalage de deux éléments, sans besoin de déclarer à l'avance de dimension. A partir du moment où elle se voit affecter deux valeurs, la variable devient automatiquement un étalage. Ainsi, la multiplication par 3 de chaque élément de Z se fait en une ligne, comme indiqué ci-bas. On appelle une variable sans dimension, un scalaire. Pour substituer les résultats dans une autre étalage variable, il suffit d'additionner les deux variables ensemble.

Pour obtenir un élément individuel d'un vecteur, vous pouvez le sélectionner grâce aux parenthèses carrées [et] et travailler avec la variable comme avec une variable sans dimension.

APL vous permet de travailler, soit avec un étalage complet, soit avec un ou plusieurs de ses éléments en particulier.

"APL traite l'information en bloc."
—— Aucun besoin de déclarer de dimension.

これらをまとめてやるには、どうしたらよいでしょうか。ここからが、APLが他の言語と特にちがうところです。

この場合、変数Zには2つの数値を入れています。つまりZは2つの要素からなる配列として機能します。しかし'DIMENSION'などと宣言することは必要ありません。2つ以上の数値が代入されると、自動的に配列として扱われます。これをそれぞれ3倍するには、まとめて行います。

ちなみに以前の1つだけのデータはスカラーと呼ばれます。

今の結果を別の配列変数に代入します。またZとWとをまとめてそれぞれ加えるのは、ごく簡単です。

配列の個々の要素を扱うには、[と]とでとり出します。そしてふつうに演算できます。

APLでは、配列データの変数を、まとめて代入したり、それぞれをまとめて演算することが極めて容易にできます。

"APLはデータをまとめて配列として扱う"
——配列の宣言はいらない。

对于同时进行的以下的运算，（是APL与其它语言特殊的不同之处。）

假设两个数赋值给变量Z，换句话说，Z是由两项元素组成的数组，但是不需要用'DIMENSION'等来予先定义它。当两个以上的数赋值给同一个变量时，这个变量自动地变成数组。一个数组的每一项乘以3可以一次完成。顺便指出，只含有一个数的变量称为标量。把结果代入另一个数组变量，或者把数组Z和数组W的各项分别相加也非常简单。

可以用[和]来指明数组的单个的项并对其进行通常的运算。

用APL能对数组的各项进行集体赋值或进行任何数学运算。

"APL对数组的数据可以集体处理不需先定义数组。"

```
      N←ι6
      N
1 2 3 4 5 6
      M1←ι6 +4
      M1
1 2 3 4 5 6 7 8 9 10
      M2←ι4 + 6
      M2
1 2 3 4 5 6 7 8 9 10
      M3←4 + ι6
      M3
5 6 7 8 9 10
      X←3×4-2
      X
6
      Y←(3×4)-2
      Y
10
```

You're introduced to the convenient function ι (iota) representing a series of numbers. Collective data from one to a given number may be generated by the function ι. By the way, all of APL processing works via functions, so that APL is called a functional language. Various examples of series are formed in such a way.

Here, M1 equals M2, but does not equal M3. The reason comes from the rule, "operations of APL should be performed from right to left." In the cases of M1 or M2, an addition is first carried out, resulting in 10, then a sequence is generated for its result. On the other hand, M3 is obtained after added 4 to a sequence of 1 to 6, respectively. So, M3 is different from M1 or M2.

Such a rule applies to an ordinary arithmetic operation. In APL arithmetic, first 4−2=2, then 3 times to it, resulting in X=6. Observe the different result of conventional arithmetic, where 3×4=12, then 12−2=10. However, with parentheses, (and), you can force doing arithmetic in conventional fashion.

"APL arithmetic is done from right to left."

Jetzt sollen Sie die nützliche Funktion ι (Jota) kennenlernen, die eine Folge von Zahlen repräsentiert. Durch die Funktion ι wird eine Datenmenge von 1 bis zur angegebenen Zahl generiert. Nebenbei, APL führt alles in Funktionen durch, deshalb wird APL eine funktionale Sprache genannt. Verschiedene Beispiele von Reihen werden so erzeugt.

M1 entspricht hier M2, jedoch nicht M3. Der Grund dafür ist die Regel 'APL arbeitet von rechts nach links'. Im Falle von M1 und M2 wird zuerst die Addition durchgeführt, das ergibt 10, dann wird für dieses Ergebnis die Zahlenfolqe erzeuqt. M3 dagegen wird errechnet, indem die Konstante 4 auf jedes Element der Zahlenfolge 1 bis 6 addiert wird. Deshalb ist M3 verschieden von M1 oder M2.

Daher hat diese Regel ganz allgemein Auswirkung auf das Rechnen. In APL-Arithmetik wird zuerst 4−2=2 gerechnet, dann mit 3 mal genommen, das ergibt X=6. Beachten Sie, dass die konventionelle Arithmetik ein anderes Resultat liefert, sie rechnet 3×4=12, dann 12−2=10. Durch das Setzen von Klammern können Sie jedoch eine Berechnung nach konventioneller Arithmetik erzwingen.

'APL arbeitet von reshts nach links.'

Nous présentons maintenant la fonction ι qui crée une suite de nombres. La suite de 1 à nombre indiqué est générée par ι. Il faut noter qu'APL fonctionne strictement par fonctions, donc on l'appelle un langage de programmation fonctionnel. Voyez les exemples donnés au bas.

Dans notre exemple, M1 et M2 ont la même valeur. Par contre, M3 est différent car, en APL, les opérations arithmétiques sont faites de droite à gauche. Pour M1 ou M2, l'addition est d'abord faite, et puis la suite est générée. Par contre, M3 est obtenu en additionnant 4 à la suite de 1 à 6 générée auparavant. Donc, M3 se voit assigner des valeurs bien différe de M1 ou M2.

Cette règle de droite à gauche s'applique à toutes les opérations arithmétiques. Notez l'exemple final où, d'abord 4−2=2, puis multiplié par 3, nous donne 6. Prenez en note cette différence importante avec l'arithmétique conventionnelle, où 3×4=12, et 12−2=10.

Heureusement, grâce aux parenthèses, vous pourrez forcer les opérations à s'effectuer dans l'ordre conventionnel.

"L'arithmétique de l'APL est effectuée de droite à gauche."

下面介绍能产生连续整数数列的方便的函数ι(iota)。它能生成从1开始到任意指定的自然数。顺便指出，所有的APL的处理数据过程都是通过函数来实现的，因此APL也叫做函数型语言。例举的各种数列都是这样生成的。

这里M1等于M2，但不等于M3。这是因为运算规则——"APL的运算是从右到左"而来的。对于M1或M2先进行加法，得到10，然后再生成数列。而M3则是对1到6的数列分别加4而得来的，因此，M3和M1或M2产生不同的结果。

这种运算规则也适用于一般的算术运算。按APL运算，首先4－2＝2，然后乘3得6。而按普通算术运算应为3×4＝12，再12－2＝10，结果是不一样的。但是，使用括号(和)可以强使之按普通算术运算方式进行。

"APL的算术运算从右至左进行"

次に、連続した整数をまとめて表すのにι(イオタ)という便利な関数があります。ι(イオタ)を用いると1からの自然数を生成できます。ところで、APLでのすべての処理は関数として働き、その意味でAPLは関数型言語と呼ばれます。これによりいくつかの自然数列を要素とする配列の変数を作ってみます。ここでM1とM2とは等しくてもM3とは等しくありません。

それは"APLの演算は'右から左へと'と行われる"という演算の順序のためです。M1とM2との場合は、まず足し算が行なわれ、その結果の10に対して、ι(イオタ)により1から10までの数が生成します。

一方、M3の場合はまずι6により1から6までの数が生成し、それに対して、それぞれ4が加えられます。したがってM3はM1やM2とは異る結果になります。

これはふつうの四則演算でも同様です。APLの演算では、まず4−2=2となり、次に3が掛けられ、Xは6となります。ふつうの計算式のように3×4=12、12−2=10とはなりません。どうしても、左から先に演算したいときは、Yのようにカッコ(と)とをつかいます。十分注意して下さい。

"APLの演算は、右から左へと行われる。"

```
      ⍴M1
10

      ⍴M2
10

      ⍴M3
6

      M1 + M2
2 4 6 8 10 12 14 16 18 20

      M1 + M3
LENGTH ERROR
      M1+M3
       ∧

      +/M1
55

      (+/M1) ÷ (⍴M1)
5.5
```

What should you do if you do not know how many elements an array is formed of? The function ρ (rho) is the thing to use. As you see, the element count is 10 for M1 and M2, while 6 for M3.

Note that an arithmetic operation is valid only to arrays each of whose elements number the same. For example, the addition M1+M2 is valid, while the addition M1+M3 is impossible, resulting in an error.

Now, let's work the operations with / (slash), one of the most eminent features of APL. It is grammatically not a function but an operator. As you see in the examples, it works like additive signs (+) inserted between the items. Namely, the addition

1+2+3+4+5+6+7+8+9+10

is performed, resulting in 55.

Utilizing this method, you may easily get an average value.

"APL works in functions and operators."

Was können Sie tun, wenn Sie nicht wissen, aus wievielen Elementen ein Bereich besteht? Nun, dafür gibt es die Funktion ρ (rho). Wie Sie sehen hat M1 und M2 10 Elemente, M3 dagegen 6.

Beachten Sie, dass jede arithmetische Operation nur für Bereiche mit der gleichen Anzahl Elemente gültig ist. Die Addition von M1 und M2 zum Beispiel ist gültig, die addition von M1 und M3 ist nicht möglich, sie ergibt einen Fehler.

Jetzt wollen wir mit / (Schrägstrich) arbeiten, einer der hervorragendsten Möglichkeiten von APL. Präzise ausgedrückt handelt es sich hierbei nicht um eine Funktion sondern um einen Operator. Wie Sie an den Beispielen sehen, arbeitet +/ so, als würde man Pluszeichen (+) zwischen die einzelnen Elemente setzen. Tatsächlich wird die Addition

1+2+3+4+5+6+7+8+9+10

durchgeführt und ergibt 55.

Auf diese Art können Sie leicht den Durchschnitt ermitteln.

'APL arbeitet mit Funktionen und Operatoren.'

Oue faire si vous ne connaissez pas le nombre d'éléments de votre étalage? La fonction ρ vous permet de connaître la longueur d'un vecteur. Par exemple, ici, M1 et M2 contiennent 10 éléments, alors que M3 n'en contient que 6.

Notez que les opérations arithmétiques entre étalage ne sont valides que si les vecteurs sont de même taille. L'addition M1+M2 sera valide, alors que M1+M3 produira un message d'erreur.

Passons maintenant à l'opération (barre penchée), une des fonctions les plus appréciées d'APL. Grammaticalement, elle n'est pas vraiment une fonction mais plutôt un opérateur. Comme illustré par notre exemple ci-bas, il permet de faire l'addition de tous les éléments d'un étalage. En d'autres mots, il remplace

1+2+3+4+5+6+7+8+9+10,
Pour obtenir le résultat de 55.

Grâce à cette opération, vous pourrez facilement obtenir une moyenne arithmétique.

"APL travaille avec de fonctions et de opérateurs."

如果想知道一个数组的项的数目该怎么办？函数ρ(rho)正是为此而准备的。可以看到M1和M2的项数为10，而M3的项数为6。

注意，只有项数相同的数组之间的算术运算才是有效的。例如M1＋M2是有效的，而M1＋M3则无效，其结果为错误。

现在让我们来看一下带有／(斜杠)的运算符，这是APL最突出的特征之一。从语法上这不是一个函数而是一个运算符。从例中可见其作用好像在数组的每项之间插入加号(＋)，即执行

1＋2＋3＋4＋5＋6＋7＋8＋9＋10
结果为55。

用这个方法计算平均值很容易。

"APL通过函数和运算符处理数据"

配列の要素の数がわからなくなってしまったときは、どうすればよいでしょう。このためには、ρ(ロー)という関数が用意されています。

M1とM2の要素の数は10、M3の要素の数は6ということです。ここで要素の数が等しい配列に対してのみ演算が行われるということにも注意して下さい。

例えば、M1+M2では結果が求められるのに対して、M1+M3のように要素の数が異なるときはエラーとなります。

ここでAPLでの最も便利な機能の1つとして、／(スラッシュ)を使った演算をとりあげます。これは関数ではなく、作用素と呼ばれるものですが、この例では、足し算+をそれぞれに対して行うような働き、すなわち各要素の間に+があるものとして働きます。つまり、

1+2+3+4+5+6+7+8+9+10
の演算が行われます。従って55という結果が得られます。

これを用いると、平均値の計算は先のρもいっしょに使っていとも簡単に行えます。

"APLには関数と作用素とがある。"

```
      M←ι12
      M
1 2 3 4 5 6 7 8 9 10 11 12

      A←3 4ρM
      A
 1  2  3  4
 5  6  7  8
 9 10 11 12
      4 3ρM
 1  2  3
 4  5  6
 7  8  9
10 11 12
```

Bisher haben Sie mit Daten gearbeitet, die linear in einer Richtung angeordnet waren und als Vektoren bezeichnet werden.

Als nächstes wollen wir uns mit Daten beschäftigen, die in zwei Richtungen angeordnet sind, nämlich in Form einer Matrix. Man nennt dies mehrdimensional.

Eine Matrix können Sie so leicht handhaben, wie Sie hier sehen. Zuerst erzeugen Sie einen Vektor von 1 bis 12, dann formen Sie ihn um in eine Matrix mit 3 Zeilen und 4 Spalten.

Jetzt lernen Sie die beiden verschiedenen Anwendungsmöglichkeiten der Funktion ρ (rho) kennen. Sie erinnern sich vielleicht, dass wir die Funktion ρ benutzt haben, um die Anzahl Elemente eines Vektors zu finden. Im Augenblick verwenden wir die Funktion ρ um die Dimension einer Matrix festzulegen, das heisst, wir verwenden ρ anders als vorhin. Sie müssen wissen, dass das gleiche Symbol zwei verschiedene Bedeutungen hat, in Abhängigkeit von der Anzahl Argumente, die der Funktion mitgegeben werden.

Eine Funktion mit einem Argument wird monadische Funktion genannt. ρ als monadische Funktion bringt als Ergebnis, wie ein Bereich strukturiert ist, nämlich die Werte, die die Form des Bereichs widerspiegeln.

Eine Funktion mit zwei Argumenten wird dyadische Funktion genannt. ρ als dyadische Funktion gibt an, wie ein Bereich generiert werden soll. Man kann damit z.B. eine Matrix mit beliebig vielen Zeilen und Spalten anlegen.

So far you've been doing things with the data arranged linearly in one direction, denoted as a vector.

Next, let's go to the data ranging in two directions, namely, in a form of matrix, which is denoted as a two -dimensional array. You can handle an array very easily as you will see. First, you get a vector of 1 through 12, then reform it into a matrix of 3 rows by 4 columns.

Now, you're learning about two different uses of the function ρ (rho).

You may remember the purpose of the function ρ was to find the number of elements composing a vector. However, the function ρ here reforms the shape of an array, which is different from the former ρ. You should notice that the same symbol ρ means two ambivalent operations, depending upon the number of arguments the function takes.

A function taking one argument is called a monadic function. ρ as a monadic function returns the data on how an array is formed, namely, the values representing a shape of an array. A function taking two arguments is called a dyadic function. ρ as a dyadic function specifies how an array is to be generated. It enables making a matrix array with any rows and columns as you will. Thus, ρ as a dyadic function is called the reshape function.

"An APL function works differently according to the number of arguments, i.e. monadic or dyadic."

Jusqu'ã maintenant, nous n'avons mentionné que les variables unidimensionnelles, les vecteurs. Passons maintenant aux variables ã deux dimensions, les matrices. On peut travailler avec des matrices tout aussi simplement qu'avec des vecteurs. D'abord, nous obtenons une suite de 1 ã 12. Et puis, on la transforme facilement en une matrice ã 3 rangées et 4 colonnes.

Vous remarquerez l'utilisation différente que l'on fait, ici, de la fonction ρ. Comme vous vous en souviendrez, ρ nous permettait d'obtenir la taille de notre vecteur. Ici, par contre, la fonction ρ nous permet de restructurer la forme de notre variable d'un étalage vecteur en une matrice. Il est donc important de noter que la fonction ρ produira des resultats différents selon le nombre d'arguments présentés.

Une fonction ne demandant qu'un argument est appellée monadique, alors qu'une fonction ã deux arguments est dite diadique. ρ, en tant que fonction monadique, nous renseigne sur la longueur et la structure de la variable. Par contre, ρ, en tant que fonction diadique, spécifie la structure de la variable. Ainsi utilisée, on la surnomme la fonction de restructuration.

"Chaque fonction APL produit des résultats différents selon le nombre d'arguments employés, c-a-d monadique ou diadique"

到此我们只涉及了一维数组，也称为矢量。

下面让我们讨论一下二维数组，即矩阵。

你将会看到处理数组是很容易的。首先生成从 1 到 12的矢量，然后再用前面所示的 ρ 把它变成 3 行 4 列的矩阵。

现在你要注意 ρ 函数的两种不同的用法。你也许会记得 ρ 函数原来的作用是得出一个矢量的项数，但现在的 ρ 函数则是改变一个数组的维数，这是和原来的作用不同的。请注意同一个符号 ρ 意味着两种不同的运算，到底哪种运算由 ρ 函数的参数数目来决定。

ρ 的右侧只有一个参数的 ρ 函数叫做单参函数， ρ 作为单参函数时给出一个数组的形状；有两个参数的函数叫做二参函数， ρ 作为二参函数时定义如何生成一个数组，它能生成有任意行和任意列的矩阵，因此做为二参函数的 ρ 也叫做变形(reshape)函数。

"根据参数目的多少，APL函数的作用不同，可以是单参函数，也可以是二参函数。

これまでの配列は、数値が一方向だけ並んだもので、これはベクトルと呼ばれます。

次に数値が二つの方向に並んだマトリクスも簡単に扱えることを見てみます。これは 2 次元の行列にほかなりません。

マトリクスは、APL では次のように操作されます。まず、1 から 12 までのベクトル M をつくります。次にこの M をもとに、3 行×4 列のマトリクス A が、前に出てきた ρ を使って作られました。

ここで、ρ という関数の 2 通りの使い方に注意する必要があります。先に出てきた ρ は、ベクトルの要素の数をしらべるものでした。今度の ρ は、これとは違いマトリクスの行数、列数をきめるものです。

この際はっきりと区別しておかなければならないことは、関数の引数のとり方により、同じ APL の文字でも違った機能をもつ、ということです。

まず引数が関数の右側に 1 つだけある場合、これは 1 項関数と呼ばれます。ρ の 1 項関数は、配列の要素の構成、つまり配列の形(shape)を表す数値を返します。

次に引数が関数の右側と左側の両方に 2 つある場合、これは、2 項関数といわれます。2 項関数としての ρ は、これから作成しようとする配列の構成を示します。これにより、望みの行数、列数のマトリクスが作られます。この意味で、ρ の 2 項関数は、変形(reshape)と呼ばれます。

"APL には引数によって 1 項関数と 2 項関数とがあり、その働きは異なる。"

```
      A[2;3]
7

      A[2;]
5 6 7 8

      A[;2 3]
 2  3
 6  7
10 11


      +/A
10 26 42

      +⌿A
15 18 21 24
```

An element of a matrix is manipulated by an index.

For example, A[2;3] represents a single element in row 2 column 3 for the array A.

Further, APL can specify all the elements of any row or column, that is a unique advantage over other languages. A[2;] represents all the elements of row 2, and A[;2 3], all the elements of columns 2 and 3.

The operator /, as already known, may be applied to a matrix. +/ does an addition of the elements of each column along the row. Another new symbol adds the elements of each row along the column.

"APL data, generally an array, is classified as a scalar, a vector or a matrix."

Ein Element einer Matrix wird durch einen Index angesprochen. So bedeutet z.B. A[2; 3] ein einzelnes Element der Matrix A in Zeile 2, Spalte 3.

Ausserdem kann APL alle Element einer beliebigen Zeile oder Spalte ansprechen, das ist ein einmaliger Vorteil gegen-über anderen Sprachen. A [2;] steht für alle Elemente der Reihe 2, A[;2 3] für alle Elemente der Spalten 2 und 3.

Der bereits bekannte Operator / gilt auch für Matrizen. +/ addiert die Elemente aller Spalten für jede Zeile. Ein weiteres neues Symbol addiert die Elemente aller Zeilen entlang den Spalten.

"APL-Daten, ganz allgemein Bereiche, werden in Skalare, Vektoren und Matrizen eingeteilt."

Un élément d'une matrice peut être accédé en spécifiant son index. Par exemple, A[2; 3] représente l' élément de la matrice A, situé à la rangée 2, et à la colonne 3.

De plus, APL offre la capacité d'accédera tous les éléments d'une même rangée, ou d'une même colonne d' un seul coup, ce qui est unique à ce langage. A[2;] représente tous les éléments de la rangée 2, et A[;2 3], tous les éléments des colonnes 2 et 3.

L'opérateur/, déja mentionné, peut être appliqué aussi pour une matice. +/ peut obtenir la somme des éléments de chaque rangée. Un nouveau symbole fait l' addition des éléments, colonne par colonne.

"L'information APL, généralement classifieé comme étalage, peut être aussi classifiée comme scalaire, vecteur ou matrice."

マトリクスの要素は、インデクス(指標とも呼ばれる)により指定されます。つまり、A [2;3] により配列Aの2行3列の1つの要素を表わします。

さらにAPL独特の表示法として、行ごと、列ごとにまとめて指定することができます。A[2;]は2行目の要素を示します。またA[;2 3] は2列目と3列目の要素を示します。

ここで先に出てきた作用素／はマトリクスにも適用できます。+／は行に沿って各列のそれぞれの要素の足し算を行った結果となります。一方、列に沿って各行のそれぞれの要素の足し算を行うには、別の記号の作用素があります。

“*APL*のデータは一般には配列であり，その種類には，スカラー，ベクトル，マトリクスがある。”

矩阵的一项数据由下标(index)指定，例如，A[2;3]表示数组A的第二行第三列的数据。APL甚至还能指定任意行或任意列的所有的项，这是优于其它语言的。A[2;]表示第二行所有的项，A[;23]表示第二列和第三列的所有的项。

前面讲过的算符/也能用于矩阵、它能使每一行中的各项相加。

可使每一列中的各项相加。

“APL的数据通称数组，分为标量(scalar)、矢量(vectar)和矩阵(matrix)。

```
      A←2 2⍴2 ¯3 5 2

      B←6 8

      A
 2 ¯3
 5  2

      B
6 8

      X←B ⌹ A

      X
1.894736842 ¯0.7368421053

      X←10 15 20 25 30

      Y←1003 1005 1010 1008 1014

      X, [0.5]Y
  10   15   20   25   30
1003 1005 1010 1008 1014

      X, [1.5]Y
 10 1003
 15 1005
 20 1010
 25 1008
 30 1014


      COEF←Y ⌹ 1,[1.5]X

      COEF
998 0.5

      YY←COEF +.× ⍉ 1,[1.5]X

      YY
1003 1005.5 1008 1010.5 1013
```

You will surely realize APL is very powerful, when you look at an operation involving the 'domino' function.

Supposing A is an array of 2 rows by 2 columns and B is a vector of 2 elements, the simultaneous linear equations

$2x_1-3x_2=6$

$5x_1+2x_2=8$

may be solved in a single operation through the 'domino' function.

A 'domino' function also may do a least square approximation, alternatively known as a regression calculation. If the y are measured as experimental values against the variable x, the regression coefficients a and b for

$y=a+bx$

are calculated as the elements of COEF, by use of the 'domino' operation. The approximated values are obtained as YY through another APL operator, inner product.

APL can do higher-level data processing through the combination of functions and/or operators, without any need of programming.

"APL can do higher-level data processing with a simple operation."

Die Mächtigkeit von APL wird deutlich, wenn man die Wirkungsweise der 'Domino'-Funktion betrachtet.

Angenommen, A sei eine Matrix von 2 Zeilen und 2 Spalten und B ein Vektor von 2 Elementen, dann können die entsprechenden linearen Gleichungen

$2x_1-3x_2=6$

$5x_1+2x_2=8$

in einer einzigen Rechenoperation mit Hilfe der 'Domino'-Funktion gelöst werden. Die 'Domino'-Funktion kann auch Approximationen nach der Methode der kleinsten Quadrate durchführen, bekannt als Regressionsanalyse. Wenn y Messzahlen zur Variablen x enthält, so werden die Regressionskoeffizienten a und b für

$y=a+bx$

als Element von COEF mit Hilfe von 'Domino' berechnet. Die Näherungswerte erhält man als YY durch einen weiteren APL-Operator, das 'Innere Produkt'.

Durch die kombination von Funktion en und/oder Operatoren kann.

APL Datenverarbeitung auf höchstem Niveau durchführen, ohne dass überhaupt programmiert werden muss.

'APL macht mit einem einfachen Befehl Datenverarbeitung auf höchstem Niveau.'

Vous réaliserez sûrement toute la puissance d'APL quand vous utiliserez la fonction domino.

Soit A, une matrice de 2 rangées et 2 colonnes, et B, un vecteur de 2 éléments, les équations linéaires simultanees

$2x_1-3x_2=6$

$5x_1+2x_2=8$

peuvent être résolues en une ligne grâce à la fonction domino.

La fonction domino peut aussi résoudre une approximation de moindres carrés, aussi connue sous le nom de régression linéaire. Si X et Y sont mesurées en tant que données expérimentales en fonction de la variable X les coefficients de régression a et b de l'équation

$y=a+bx$

sont calculés en tant qu'éléments de COEF, en utilisant la fonction domino. Les valeurs approximatives sont donc obtenues en YY grâce à un autre opérateur, le produit intérieur.

APL peut réussir des manipulations de données trés avancées grâce à la combinaison de fonctions et/ou d'opération, sans besoin de beaucoup de programmation.

"*APL réussit des manipulations de données avancées à l'aide de simples opérations.*"

当你了解到多术诺函数(domino)所涉及的运算时，你一定会认识到APL的能力。

设A为2行2列数组，B为有2项的矢量，线性联立方程：

$2x_1 - 3x_2 = 6$

$5x_1 + 2x_2 = 8$

可以通过多术诺函数用一步运算来解。

多术诺函数也能进行最小二乘近似法运算，也称为回归。设变量y是对自变量x的实验值，则对于

$y = a + bx$

的回归素数a和b可通过多术诺运算做为COEF的项而算得，而最小二剩法的近似值y则通过另一个APL运算(内积)做为YY而求出。

象这样，APL通过函数和操作符的组合，就能进行高级的数据处理，而不需编程序的工作。

"APL能用简单操作进行高级数据处理"

APLの強力さを示すものとして'ドミノ'と呼ばれる関数の使い方を示しましょう。

Aを2行2列のマトリクス、Bを2つの要素のベクトルとするとき、連立一次方程式

$2x_1-3x_2=6$

$5x_1+2x_2=8$

の解は、関数'ドミノ'を用いた1回の操作で求められます。

また、関数'ドミノ'は、最小自乗近似とも呼ばれる回帰式の計算が簡単に行なえます。変量xに対する実験値としてyが測定されたとき、

$y=a+bx$

に対する回帰係数a、bは'ドミノ'を用いてベクトルCOEFの要素として得られます。また、最小自乗法の近似値は、APLの内積と呼ばれる作用素を用いてYYとして求められます。

このようにしてAPLでは関数と作用素のみでプログラムを組むことなく、高度のデータ処理ができます。

"APLは、簡単な操作で高度のデータ処理ができる。"

```
      S←'APL'

      T←'IS HAPPY'

      S, T
APLIS HAPPY

      S,' ',T
APL IS HAPPY



      U←S,' ',T
      U
APL IS HAPPY

      ρU
12


      V←6 2ρU

      V
AP
L
IS
 H
AP
PY

      V←2 6ρU

      V
APL IS
 HAPPY
```

How then is a character string used in APL? It is as easy to use for this as a pocket calculator.

A character string is bundled with quotation marks, freely substituted into S and T. Needless to say, no declaration is needed.

Do this with simple examples. Joining the strings S and T is easily carried out with , (comma). Alas! you got the string with no space character, so insert a space character between them and substitute it into the variable U.

How do you imagine such string data is actually composed? Let's check this out by ρ. You will find that a character string is formed of as array whose elements are characters. It follows that a character string may be variously reshaped by ρ.

"APL treats character string data as a character array."

——No declaration for a character array is needed.

Wie ist das nun mit einer Zeichenkette in APL? Nun, das ist ebenfalls so einfach, wie einen Taschenrechner zu benutzen.

Eine Zeichenkette wird in Hochkommata eingeschlossen. In unserem Fall haben wir die Zeichenkette geteilt und S und T zugewiesen. Es ist überflüssig zu sagen, dass auch hierfür keinerlei Deklarationen nötig sind.

Einige einfache Beispiele genugen. Das Aneinanderfügen der Zeichenketten S und T erfolgt einfach mit ,(Komma). Ach, Sie haben die Zeichenkette ohne Leerzeichen erhalten? Fügen Sie einfach ein Leerzeichen zwischen die beiden Variablen und weisen Sie das Ergebnis der Variablen U zu.

Wie stellen Sie sich den Aufbau einer solchen Zeichenkette vor? Lassen Sie uns probieren, wie es mit dem ρ steht. Sie werden sehen, dass eine Folge von Zeichen als ein Bereich dargestellt wird, dessen Elemente die Zeichen sind. Daraus folgt, dass eine Folge von Zeichen mit ρ beliebig umstrukturiert werden kann.

"APL behandelt eine Zeichenfolge als Zeichen -Bereich."

——Eine Deklaration für einen Zeichen-Bereich ist nicht nötig.

Comment travailler avec des chaînes de caractères? C'est aussi facile qu'avec une calculatrice de poche.

Une chaîne de caractères doit être définie par des guillemets, et peut être affectée à des variables tels S ou T, comme illustré dans notre exemple. Il va sans dire que, comme précédemment, nulle déclaration n'est necessaire.

Voici quelques exemples simples qui illustrent bien la manipulation de caractères en APL. La juxtaposition des chaines S et T est faite simplement à l'aide de virgules. Hélas, il manque un espace blanc à notre exemple, ce à quoi on peut facilement remédier en créant la nouvelle variable U.

Que peut bien contenir une telle chaîne? Nous pouvons facilement répondre à la question en examinant le contenu de la chaîne avec la fonction ρ.

Vous vous apercevrez qu'en fait une chaîne n'est qu'un vecteur constitué de caractères individuels. Donc, une chaîne aussi peut être restructurée avec ρ.

"APL traite une chaîne de caractères comme un é de caractères individuels."
—— Nulle déclaration n'est nécessaire.

次に、APLでは文字列の扱いについてはどうでしょうか。これも、電卓なみの手軽さで、自由自在です。

文字列は引用符でくくって、それぞれをSとTとに代入できます。もちろん文字列の変数にも宣言は一切必要ありません。

文字列の扱い方の簡単な例を行ってみましょう。文字列SとTの2つをつなげるには、ごく簡単に、(コンマ)を用いて行います。スペースなしにつながってしまいました。スペースを間に入れてつなぎ、この文字列を変数Uに代入します。

これらの文字列は実際にはどういうデータなのでしょうか。ρ(shape)を用いてしらべてみます。このように文字列のデータは、文字を要素とする文字配列になっています。したがってρ(reshape)を使っていろいろなマトリクスの形にすることもできます。

"APLでは、文字列データは、配列として扱う"
—— 文字配列に宣言は、いらない。

那么，APL中怎样处理字符串呢？这也和使用计算器一样地简单、方便。

字符串两边用引号，可以随意代入变量S和T，不需事先定义。

请做一些简单的例子。字符串可用逗号方便地连接起来。啊！你的连接后的字符串缺一个空格，请插入一个空格并把结果代入变量U。

这些字符串实际是什么样的数据呢？

让我们用ρ来检验一下，这些字符串的数据实际成了把字符作成数值的字符数组。因此使用ρ可以作成各种形式的矩阵。

"APL把字符串数据做为数组处理，不用予先定义字符数组。"

```
      C←'APL IS HAPPY '

      ρC
13

      C
APL IS HAPPY


      D←12 27ρC

      D
APL IS HAPPY APL IS HAPPY A
PL IS HAPPY APL IS HAPPY AP
L IS HAPPY APL IS HAPPY APL
 IS HAPPY APL IS HAPPY APL 
IS HAPPY APL IS HAPPY APL I
S HAPPY APL IS HAPPY APL IS
 HAPPY APL IS HAPPY APL IS 
HAPPY APL IS HAPPY APL IS H
APPY APL IS HAPPY APL IS HA
PPY APL IS HAPPY APL IS HAP
PY APL IS HAPPY APL IS HAPP
Y APL IS HAPPY APL IS HAPPY


      E←' ',[1](' ',D,' '),[1]' '

      F←'*',[1]('*',E,'*'),[1]'*'

      F
*******************************
*                             *
* APL IS HAPPY APL IS HAPPY A *
* PL IS HAPPY APL IS HAPPY AP *
* L IS HAPPY APL IS HAPPY APL *
*  IS HAPPY APL IS HAPPY APL  *
* IS HAPPY APL IS HAPPY APL I *
* S HAPPY APL IS HAPPY APL IS *
*  HAPPY APL IS HAPPY APL IS  *
* HAPPY APL IS HAPPY APL IS H *
* APPY APL IS HAPPY APL IS HA *
* PPY APL IS HAPPY APL IS HAP *
* PY APL IS HAPPY APL IS HAPP *
* Y APL IS HAPPY APL IS HAPPY *
*                             *
*******************************

      ρD
12 27

      ρE
14 29

      ρF
16 31
```

Now, it is time to explain how the pattern shown at the beginning was generated.

First, C is formed from the string data, 'APL IS HAPPY' followed with a single space character. By use of ρ, it is reshaped into the matrix of 12 rows by 27 columns, that is a unique characteristic of the ρ function. Generally ρ makes an array according to the left arguments, and items of the array are repeated over and over again, if the items specified by the right argument are not enough to form an array.

In this way, you get a character array D, which represents the pattern. Then, you join space characters around D, before and after, right and left, resulting in a character array E. Finally, you finish wish ' * ' around E, before and after, right and left.

Now, you've successfully got the pattern, whereby you appreciate such short coding.

Jetzt wird es aber Zeit zu erklären, wie das Muster, das wir zu Beginn gezeigt haben, erzeugt wird.

Zuerst bilden wir C aus der Zeichenkette 'APL IS HAPPY'. gefolgt von einem Leerzeichen. Mit Hilfe von ρ wird es in eine Matrix mit 12 Zeilen und 27 Spalten umgeformt. ρ erzeugt generell einen Bereich entsprechend dem linken Argument und die Elemente des Bereichs werden immer wieder wiederholt, wenn die Elemente des rechten Arguments nicht ausreichen, den Bereich zu füllen.

Auf diese Weise erhalten Sie den Zeichenbereich D, der das Muster darstellt. Dann fügen Sie Leerzeichen rund um D ein, davor und dahinter, rechts und links, daraus ergibt sich der Zeichenbereich E. Schliesslich füqen Sie noch das Zeichen ' * ' rund um E ein, davor und dahinter, rechts und links.

Jetzt haben Sie erfolgreich das Muster erzeugt, und dabei festgestellt, wie gering der Programmieraufwand ist.

Maintenant, il est grand temps de vous expliquer comment l'image du début se génère.

D'abord, la chaîne de caractères 'APL IS HAPPY', suivie d'un espace blanc, est affectée à la variable C. Avec ρ, C est restructurée en une matrice de 12 rangées par 27 colonnes, ce qui illustre la nature unique de ρ. Généralement, ρ restructure un étalage en une matrice en répétant le vecteur autant de fois que nécessaire, si les éléments spécifiés ne sont pas assez nombreux pour former la matrice désirée.

De cette façon, vous obtenez la matrice de caractères D, qui forme une partie de l'image voulue. Ensuite, vous juxtaposez des espaces blancs autour de D, pour obtenir la matrice E. Finalement, vous complétez avec des étoiles '*' tout autour de la matrice.

Et voilà l'image du début, obtenue avec seulement quelques lignes de programmation.

ここでこの手引書の最初にあげたパターンの作り方を順をおって説明してみましょう。

まずAPL IS HAPPYの最後にスペースを1つ入れて、Cのような文字列データを作ります。

これを、ρ(reshape)を用いて、12行×27列のマトリクスにするのです。ここで関数ρのマトリクス作成の際に次の性質を利用しています。文字列のデータを使って、指定したマトリクスに、1文字ずつつめていくのですが、文字が足りなくなったら、くり返すということをうまく利用しているのです。

このようにして文字配列Dとして、パターンが出来上りました。あとは前後、左右にスペースをつけて文字配列Eをつくります。さらに文字配列Eの前後、左右に'*'をつけて、文字配列Fとします。

このようにして簡単にパターンができあがります。

现在是说明怎样生成封面上的图样的时候了。

首先把'APL IS HAPPY'加上一个空格赋值给变量C，用ρ函数使之变成12行27列的矩阵，这是ρ函数的一个独特之处。一般说来ρ按照左边的参数生成数组，如果右边参数所指之数据数目不足以填满所生成之数组的话则将重复使用其内容。这样得到了数组D，它代表了所示图样。然后在D的前、后、左、右加上空格。得到E。最后再在E的四周加上'*'得到F，就完成了整个图样，非常简单。

```
        ∇ TEST1 X
[1]     'TOTAL:'
[2]     +/X
[3]     'AVERAGE:'
[4]     (+/X) ÷ (⍴X)
[5]     ∇

        A←123.45 ¯0.25 7.02 0 60.38

        TEST1 A
TOTAL:
190.6
AVERAGE:
38.12

        B←⍳100

        TEST1 B
TOTAL:
5050
AVERAGE:
50.5
```

The usage explained so far is called a command mode or an execution mode, where you key-in and immediately afterwards execution occurs. In defintion mode, another usage of APL, you define your own function as you will, and run it, so-called making a program.

Execution mode (command mode)

Definition mode (edit mode)

You can do a fair amount of processing just by keying-in under execution mode. Further, more complex processing can be achieved by making a program in definition mode.

Entering the ▽(del) symbol followed by a function name, you're going into definition mode. When entering ▽ again, you return to execution mode.

Let's make a function called as TEST1. The system types out a line number such as [1], then you type in 'TOTAL:' at the cursor. Continuing, you type in +/X for [2]. Finally typing ▽ in, you get a defined function, the TEST1.

In advance, putting data into a variable A or B as the argument of the function, you may run it just by entering the function name.

Once you define a function, you may do your desired processing by using it in execution mode. There is no difference between an originally-installed function called a primitive function and a user-defined function. While executing a programmed function, you need no operation such as "RUN TEST1" required by other languages.

"APL enables user to execute his/her desired function as a program."

Now, you've mastered how to use APL at the introductory level.

Die Nutzung von APL, wie sie bisher erklärt wurde, nennt man den Befehlsmodus oder Ausführungsmodus, wobei unmittelbar auf die Eingabe die Ausführung folgt. Der Definitionsmodus, eine weitere Anwendungsform von APL, bedeutet, dass Sie Ihre eigene Funktion definieren wie Sie möchten und sie laufen lassen, das heisst, ein Programm schreiben.

Ausführungsmodus (Befehlsmodus)

Definitionsmodus (Eingabemodus)

Sie können eine recht grosse Menge an Verarbeitung durchführen, nur indem Sie im Befehlsmodus Ihre Eingaben machen. Sie können aber ausserdem eine viel komplexere Verarbeitungen erzielen, indem Sie ein Programm, im Definitionsmodus erstellen.

Wenn Sir ▽ (Nabla), gefolgt von einem Funktionsnamen eingeben kommen Sir direkt in den Definitions modus. Wenn Sie dann später wieder ▽ eingeben, so kommen Sie wieder in den Ausführungsmodus zurück.

Lassen Sie uns eine Funktion erstellen, die wir TEST1 nennen wollen. Das System gibt eine Zeilennummer an, z.B. [1], dann geben Sie dort etwas ein, wo der Cursor steht, z.B. 'TOTAL:'. Wenn Sie die Entertaste gedrückt haben, geben Sir in Zeile [2] ein: +/X usw. usw. Wenn Sie schliesslich ▽ eingeben, erhalten Sie eine definierte Funktion, in diesem Falle TEST1.

Sobald Sie den Variablen A und B, die Sie dann als Argumente benutzen, Daten zugewiesen haben, können Sie die Funktion laufen lassen, indem Sie nur den Funktionsnamen eingeben.

Sobald Sie eine Funktion definiert haben können Sie die gewünschte Verarbeitung durchführen, indem Sie die Funktion im Ausführungsmodus aufrufen. Es gibt keinen Unterschied zwischen den ursprünglich installierten Funktionen, genannt 'Primitive Funktionen' und den vom Benutzer definierten Funktionen. Wenn Sie eine selbst definierte Funktion laufen lassen brauchen Sie keinen besonderen Befehl, wie z.B. ' RUN TEST1' wie in anderen Programmiersprachen.

'APL ermöglicht es dem Benutzer, seine/ihre gewünschte Funktion als Programm auszuführen.'

So, jetzt haben Sie es geschafft. Sie wissen, wie man APL nutzt - APL für Anfänger.

Jusquà maintenant, nous n'avons travaillé qu'en mode de commande, appellé aussi mode d'exécution. C'est ce mode qui produit les résultats immédiatement après que vous les ayez entrés. Un autre mode d'usage en APL, le mode de définition, vous laisse déclarer vos propres fonctions et vous laisse les appeler comme dans un programme.

Mode d'exécution (mode de commande)
Mode de définition (mode d'édition)

Vous pouvez faire un grand nombre de manipulations, seulement avec le mode d'exécution. Par contre, des opérations encore plus complexes peuvent être réussies grâce au mode de définition.

Il suffit d'entrer le symbole ▽ (del), suivi d'un nom de fonction pour passer en mode d'édition. Un nouveau ▽ vous permettra de revenir en mode de commande.

Essayons, en créant une fonction nommée TEST1. Le système indiquera un numéro de ligne, tel que [1], et vous pouvez continuer en inscrivant 'TOTAL:'. A la ligne [2], écrivez +/X et ainsi de suite comme l'indique notre exemple ci-bas. Finalement, en entrant de nouveau un ▽, vous obtenez une fonction définie, TEST1. Il suffit d'initialiser A ou B, utilisé comme argument, et vous pouvez invoquer votre fonction comme illustré ci-bas.

Une fois la fonction définie, vous pouvez l'utiliser dans vos calculs en mode d'exécution comme n'importe quelle autre fonction vue précédement. Il n'y a aucune différence entre une fonction prédéfinie, appellée fonction primitive, et une fonction définie par l'usager. Au moment de l'exécution, vous n'avez pas à entrer 'RUN TEST1' comme pour d'autres langages.

"APL permet aux usagers d'exécuter leurs propres fonctions comme des programmes"

Et voilà. Vous maîtrisez maintenant l'utilisation élémentaire d'APL.

これまでの使い方は、キーインすれば、即実行するということで、これはコマンドモード、あるいは実行モードといわれます。APLには、もう1つ別の使い方があります。ユーザ自身による関数を定義し、それを用いて実行するやり方です。いわゆるプログラムを作るモードです。

・実行モード(コマンドモード)
・関数定義モード(エディットモード)

APLでは、かなりの処理が実行モードとしてキーインするだけで実行できますが、さらに複雑な処理は、関数定義モードでプログラムを組んで行います。

関数定義モードへは、▽と定義したい関数の名前を入力することによって入ります。再び▽を入力すると、もとの実行モードにもどります。

簡単なユーザ定数関数を作成してみましょう。定義する関数の名前をTEST1とします。

システムからは[1]と行番号を打ちだしてくるので、これに対してカーソルの位置から、'TOTAL:'というように打ち込みます。次に[2]に対しては+／Xというように次々と入力していきます。最後に▽を入れれば、ユーザ定義の関数としてTEST1が出来上ります。

次に変数AあるいはBにあらかじめデータを入れておき、これを引数として、関数名を打ち込むだけで実行が行われます。

このように、一度ユーザが関数を定義しておけば、これを実行モードで使うだけで、ユーザの望みの処理ができます。つまり、使用する上では、これまで出て来た原始関数とよばれるもともとの関数とユーザ定義関数とに区別はありません。またプログラムを作って実行するとき、他の言語の場合のように'RUN TEST1'などとする必要もありません。

"APLでは、プログラムとしてユーザが望みの関数を定義できる。"

以上であなたはAPLがどんなものかを学んだことになります。

以上介绍的用法叫做命令模式或执行模式，键入之后立即执行。APL的另一种模式——定义模式，在定义模式下首先定义自己的函数，然后运行它，这也是所谓的编写程序。

执行模式(命令模式)
函数定义模式(编辑模式)

在执行模式，只要键入就能完成大量的数据处理工作，更复杂的过程则可以在函数定义模式下用编制程序来完成。在函数名之前先键入符号▽(del)就进入定义模式，当再次键入时就又回到执行模式。

让我们来造一个名为TEST 1的函数，系统显示行号[1]，在光标处键入"TOTAL"，再在第二行键入+/×，最后键入▽，函数TEST 1就造好了。予先给变量A或变量B赋值，再用A或B作为参数来运行这个函数。

象这样，用户定义好一个函数之后，就可以在执行模式下用它来进行期望的数据处理。原来就有的函数(叫做原始函数）和用户定义的函数之间没有区别。运行程序时不需象在其它语言中那样用"RUN TEST 1"之类的命令。

"APL可使用户把所期望的函数当作程序来运行。"

至此你已掌握了APL的初步用法。

I hope this booklet more improved. I would like you to communicate your comments about it.

Your comments:

Your name: ______________________________

company (or school): ______________________________

address: ______________________________

____

Thank you for your cooperation.

Cut here

Fold and Tape

Please
Postage
Here

To: Toshio Nishikawa
National Chemical Laboratory for Industry
Tsukuba Research Center
Tsukuba, Ibaraki 305
JAPAN

Fold and Tape